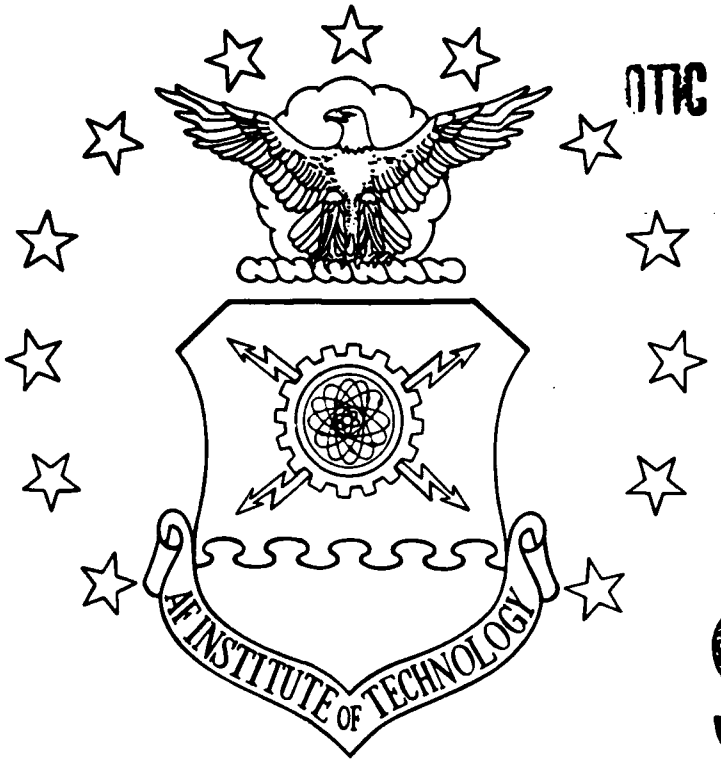

AFIT-EN-TM-87-7
Air Force Institute of Technology
Environment Portability and Extensibility Measures
Capt Scott A. DeLoach
10 August 1987

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT-EN-TM-87-7
Air Force Institute of Technology
Environment Portability and Extensibility Measures
Capt Scott A. DeLoach
10 August 1987

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT-EN-TM-87-7	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/EN	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Systems Avionics Division; 8b. OFFICE SYMBOL (If applicable): AFWAL/AAAF-2	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code): Air Force Wright Aeronautical Laboratory, Wright-Patterson AFB, Ohio 45433-6583	10. SOURCE OF FUNDING NOS.
11. TITLE (Include Security Classification): See Box 19	PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
12. PERSONAL AUTHOR(S): Scott A. DeLoach, Capt, USAF	
13a. TYPE OF REPORT: Technical Memorandum; 13b. TIME COVERED: FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day): 1987 August 10; 15. PAGE COUNT: 7
16. SUPPLEMENTARY NOTATION	

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR.	
12	05		Measurement, Computer Programming, Environments, Computer Program Verification

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: ENVIRONMENT PORTABILITY AND EXTENSIBILITY MEASURES

Advisor: Richard R. Gross, Lt Col, USAF
Assistant Dean
School of Engineering

Approved for public release: IAW AFR 190-1.
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AFIT)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED [X] SAME AS RPT. [X] DTIC USERS [ ]	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Richard R. Gross	22b TELEPHONE NUMBER (Include Area Code): (513) 255-3025; 22c. OFFICE SYMBOL: AFIT/EN

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

1.0 Introduction

To validate my thesis (DeLoach, 1987), that environments based on abstract interfaces provide enhanced portability and extensibility, I decided to compare my environment against existing non-interface based environments. To do this, I needed two metrics: one to measure portability, and the other to measure extensibility. These metrics had to be applicable to the environment as a whole and had to exhibit the following characteristics. First, the metrics had to be objective: given sufficient knowledge of the system, an individual should be able to compute only one possible measurement. Second, the metric must be applicable given only the environment design documentation and source code, thus ensuring the metrics are based solely on the characteristics of the software.

2.0 Portability Measurement

Intuitively, software (environment) portability, "the ease of movement among distinct solution environments" and "a function of the number and complexity of the requisite changes" (Yourdon and Constantine, 1978: 322), should be fairly easy to measure. There have been two approaches to measuring software portability. The first approach describes portability in terms of the effort expended to rehost the software. Although this approach measures both size and complexity, it also includes other factors such as personal skills, tool availability, etc.

The second approach is simple and straightforward. It measures the amount of code changed during the rehosting process. Although it does measure the change size and is dependent solely upon the software, it does not measure the complexity. However, an extension to the lines of code approach that considered complexity could meet the metric requirements discussed above.

Change complexity includes many components, most of which are hard to measure (e.g. understanding new host operating systems, simulating non-existent host computer functions, intrinsic difficulty of the function, etc.). However, one important aspect of complexity that is quantifiable is localization of the changes. If a piece of software requires changes to 500 lines of code, it is much easier to rehost if the changes are located in five modules instead 50. Therefore, I propose to measure the size of the change by lines of code and the complexity by locality of the changes (how many modules or packages have to be changed). Therefore, the size of a change is simply:

$$\text{size} = (1 - N_{lc}/N_{tl})$$

where N_{lc} is the number of source lines of code added, deleted, or modified, and N_{tl} is the total number of source lines. (Source lines are defined as one executable or declarative statement.)

The locality (complexity) of the portability changes is described in two parts: modularity and locality. Modularity describes the percentage of modules changed whereas locality describes the number of packages changed. Modularity is computed as:

$$\text{modularity} = (1 - N_{mc}/N_{tm})$$

where N_{mc} is the number of modules requiring change and N_{tm} is the number of total modules. Likewise, the locality of changes is computed as:

$$\text{locality} = (1 - N_{pc}/N_{tp})$$

where N_{pc} is the number of packages changed and N_{tp} is the total number of packages in the system.

The overall environment portability is then computed as the average of the individual portability metrics.

$$\text{portability} = (\text{size} + \text{modularity} + \text{locality}) / 3$$

When talking about the portability of a single component, it is usually rather difficult to talk about locality as defined above since each component probably consists of very few and probably only one package. Therefore, to measure the portability of a single environment component, only the size and modularity are used.

$$\text{component portability} = (\text{size} + \text{modularity}) / 2$$

3.0 Extensibility Measurement

Environment extensibility has been defined as the ability of an environment to adapt to major changes in the environment's capabilities and has four basic components:

Tool Modification
New Tool Integration
Existing Tool Integration
Database Extension

Although extensibility has been recognized as being a critical factor in environment success, there has been no real attempt to measure it. In the more general case, extensibility is usually considered part of software maintenance. It is also generally accepted that software structure (modularity) is critical to software maintainability (Glass, 1979: 158).

The quality of program modularity is measured by two well accepted measures: *coupling and cohesion* (Pressman, 1982: 158). Because environment extensibility is similar to normal software system extensibility (think of each environment component as a module), a measure similar to coupling and cohesion is proposed to determine environment extensibility. Therefore, I have developed four levels of extensibility to measure two basic areas: tool integration and database extension. To measure the quality of environment extensibility, each type of extensibility (tool integration or database extension) is placed into one or more of the following categories.

Indirectly Extensible
Directly Extensible
Tool Extensible
Structure Extensible

Indirect Extensibility refers to the ability to integrate a tool or data object through use of data structures external to the environment components. An analogy to this type of extensibility is indirect addressing. In indirect addressing, the program code refers to a known location that holds the desired address. This type of extensibility is powerful because it allows simple integration yet remains very flexible.

Direct Extensibility is similar to indirect extensibility in that it requires no change to the environment components themselves. In the case of direct extensibility, instead of changing an external data structure, the new tool or object is required to meet the environment's non-modifiable reference scheme (i.e. a tool must be placed in a particular directory). Although direct extensibility may require less effort than indirect extensibility, it is also less flexible. Again, the analogy for direct extensibility can be made to direct addressing where the actual address is placed directly in the program code. If the address changes, the program must be modified everywhere the address is used.

Tool Extensibility refers to a situation where to integrate a tool or database object, one or more tools must be altered. A common example of this is when a environment's command line interpreter must be changed to recognize a new tool, or, when the database tool must be changed to allow incorporation of new object types. This type of change, as a minimum, requires recompilation and retesting of all the environment components affected.

The worst type of extensibility is ***structure extensibility***. In structure extensibility, the overall structure of the environment must be changed to incorporate new tools or database objects. If the structure of

the environment changes, then the assumptions made about the environment change causing an enormous amount of redesign, retest, and reintegration of environment components. Obviously, this type of extensibility is highly undesirable.

To compute a single environment extensibility rating requires associating values with each type of extensibility discussed above. To develop a quality metric, a considerable effort would be required to determine how much better or worse any one particular type of environment extensibility is than the others. However, because exact numbers are not necessary to show a general trend, intuitive estimate for these values are used. The following equations reflect that estimate.

$$\text{tool extensibility} = 1*N_i + 5*N_d + 20*N_t + 100*N_s$$

$$\text{database extensibility} = 1*N_i + 5*N_d + 20*N_t + 100*N_s$$

$$\text{extensibility} = \text{tool extensibility} + \text{database extensibility}$$

where N_i is the number of indirect references modified, N_d is the number of direct references satisfied, N_t is the number of tools modified, and N_s is the number of structural modifications. Therefore, to apply this measure, each category of extensibility (and the number of times it applies) is counted, multiplied by its associated weight, and then summed.

References

DeLoach, Scott A. *An Interface Based Programming Support Environment*, MS Thesis - DRAFT, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, June 1987.

Glass, Robert L. *Software Reliability Guidebook*, Englewood Cliffs, New Jersey: Prentice-Hall, 1979.

Pressman, Roger S. *Software Engineering: A Practitioner's Approach*, New York: McGraw-Hill, 1982.

Yourdon, Edward and Larry Constantine. *Structured Design*, New York. Yourdon Press, 1978.